Diverse Workspace Path Planning for Robot Manipulators

Ana Huamán Quispe and Mike Stilman [1]

July 2012
GT-GOLEM-2012-003

Center for Robotics and Intelligent Machines
Georgia Institute of Technology
Atlanta, GA 30332

Abstract

We present a novel algorithm that generates a set of *diverse* workspace paths for manipulators. By considering more than one possible path we give our manipulator the flexibility to choose from many possible ways to execute a task. This is particularly important in cases in which the best workspace path cannot be executed by the manipulator (e.g. due to the presence of obstacles that collide with the manipulator links). Our workspace paths are generated such that a distance metric between them is maximized, allowing them to span different workspace regions. Manipulator planners mostly focus on solving the problem by analyzing the configuration space (e.g. Jacobian-based methods); our approach focuses on analyzing alternative workspace paths which are comparable to the optimal solution in terms of length. This paper introduces our intuitive algorithm and also presents the results of a series of experiments performed with a simulated 7 DOF robotic arm.

Keywords: Path diversity, path planning, redundant manipulators

[1]The authors are with the Center for Robotics and Intelligent Machines at the Georgia Institute of Technology, Atlanta, GA 30332, USA. ahuaman3@gatech.edu, mstilman@cc.gatech.edu

1 Introduction

Robotic manipulators have been used in industrial environments for many years. Their popularity is due in part to the fact that these manipulators are well suited for a variety of tasks ranging from spray painting to welding. There is extensive literature on the robotics community related to the most common manipulator planning problem: Given a starting configuration for the robot and a workspace target location for the end effector, generate a jointspace path such that the manipulator efficiently reaches the target location while avoiding collisions with static obstacles.

Although there exist numerous solutions to manipulator path planning, there is still not a planner for high dimensional manipulators that produces paths that are *repeatable*, *bounded in length*, *deterministic* and that *does not require pre or post-processing*. In this paper, we build on the basic concept of *Motion Rate Control* [17] and improve it to address these challenges.

Traditionally, the manipulator path planning problem was implemented deterministically as a hierarchical process where a low-dimensional global planner provided input to a higher-dimensional local planner. Such planners first find a workspace path for the end effector and then track this path, mapping it into configuration space [17, 15]. One approach to generating the initial workspace path needed is to find the *optimal workspace path* in terms of length. However, this decision may not be appropriate, leading to incompleteness.

Consider the robot arm shown in Fig.1. The task is to reach the red bottle on top of the table. We first find the shortest end effector path, which is shown in Fig.2(a). However, as we can see in Fig. 2(b), the manipulator cannot execute this path since it is out of its reach. By considering a slightly longer path, such as the path shown in Fig.2(c), we are able to successfully execute the path by mapping it into jointspace. Notice that this path is not significantly longer than the optimal one found first.

This example is one of many in which it makes sense for a planner to produce more than one workspace path to be considered during the planning process. By trading length with feasibility we produce a more robust planner. In this paper we focus on the generation of a set of diverse workspace paths to be mapped to joint space.

2 Related Work

Most recent work on path planning for manipulators has adopted the sampling-based paradigm including Probabilistic Roadmaps [12], RRT [14] and its varia-

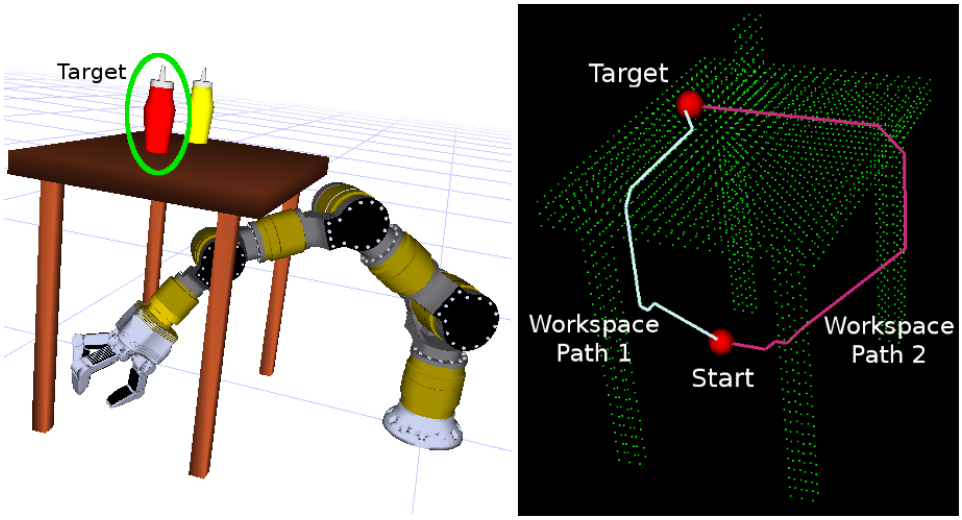
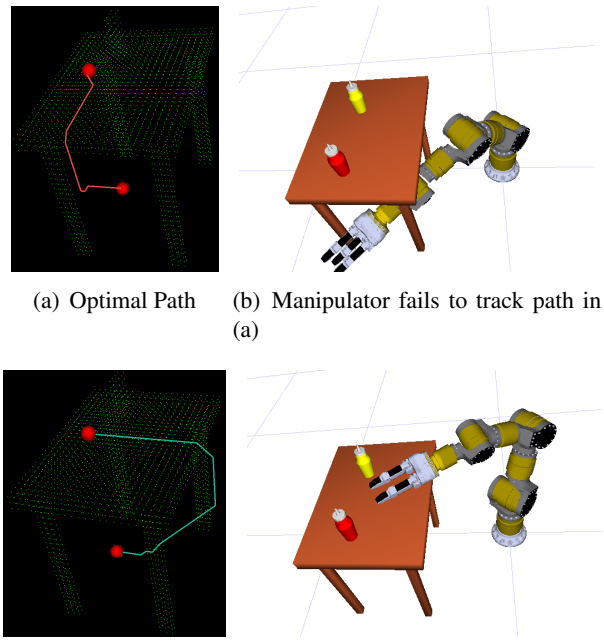


Figure 1: Problem to solve and results from our planner

tions. While these planners are not complete, they are probabilistically complete and often find solutions where workspace tracking methods do not. However, these methods are neither repeatable nor deterministic. Furthermore they require pre-processing in the case of PRM and post-processing in the form of path shortening for RRT.

We observe that such novel methods of path generation were enabled in part due to increased computational power which allowed for significantly faster nearest-neighbour queries and collision checks. We also note that such computational power can be used to quickly search and operate on 3D grids of voxels. In this paper, let us reconsider the utility of the traditional approach to manipulator path planning.

Most early planners for manipulators were inspired by the work of Whitney [17]. *Resolved Motion Rate Control*, consisted of obtaining jointspace paths from final workspace locations by applying the Jacobian pseudoinverse. Maciejewski [15] addressed the issue of obstacle avoidance by using the Jacobian nullspace to explore alternative configurations that still allowed the manipulator to follow a workspace path. Observe that these approaches focused on finding alternative configuration space paths while keeping the workspace path fixed. We approach the problem by noticing that there are cases in which the main restriction for the end effector path are the starting and goal locations, giving us flexibility to choose the trajectory to follow. We therefore choose to find alternative workspace paths that are as *diverse* as possible, so that they represent different ways to achieve a



(a) Optimal Path (b) Manipulator fails to track path in (a)

(c) Distinct Path (d) Manipulator following path in (c)

Figure 2: Illustration of the utility in considering multiple workspace paths.

task.

Previous efforts to produce *diverse* paths have largely focused on homotopy classification. Early work on homotopy in 2D spaces includes geometric-based approaches such as [7, 9] and PRM-based methods such as [16]. More recently [1] and [10] proposed methods to generate optimal paths in different homotopic classes . For the 3D scene, [2] presents an optimal planner that identify different homotopic paths. It is important to consider that in a 3D environment, such as the workspace that we consider in this paper, homotopy classification may not be useful to produce diverse paths: A scene with multiple finite obstacles can easily have only one homotopic family, which limits the number of differently classified paths.

Other alternative approaches to generate diverse paths have been proposed, each of them with different motivations. For instance in [11], Jaillet proposed *Path Deformation Roadmaps*, producing roadmaps (2D and 3D) which can be turned into different paths that maximize the relative deformation between them, making them as *diverse* as possible. The term *path diversity* has been also used by [6] to define path sets. Under this concept, a set of paths is diverse if it maximizes the *mutual separation between its paths* (also called *dispersion*). Extensions of this work applied to offline generation of local paths are presented in [13]. Our approach is similar to [6] in the sense that our algorithm produces a *sequence* of paths that tries to maximize the diversity between them. In our context, *diversity* is quantified by the accumulated distance between a path and the rest of paths in the set, not by dispersion (although these metrics are related). Furthermore, [6] assumes that the optimal set of paths is a subset P^* generated from a given, larger, set of paths ($\mathcal{X}$). In our approach, we do not require the set $\mathcal{X}$, since our algorithm only generates the paths to be considered by using graph-search techniques (such as A^* [8]) in the discretized environment.

3 Definitions and Notations

We present an algorithm to produce a sequence of k diverse workspace paths, joining the given start and target locations in a 3D static and discretized environment. In this section we will describe the search space, make the necessary definitions and formally define the *diverse path generation*.

3.1 Search Space

We consider our search space to be a bounded space discretized in voxels. A voxel is *occupied* if an object other than the robot occupies part or all of it. Otherwise

the voxel is considered *free*. For convenience, we represent this search space as an undirected connected graph $\mathcal{G}$, in which the vertices $\mathcal{V}$ are composed by the free voxels. Local connections between two adjacent free voxels are represented by edges. Notice that even when a voxel may be occupied by some part of the manipulator at some point, we do not consider it as occupied, since this is a temporary state. We only consider static obstacles as permanently occupied, so they are never included in the search space.

From the definition above, a *workspace path* P can be informally expressed as a sequence of adjacent vertices $p_i \in \mathcal{V}$ such that:

$$P = (p_1, p_2, p_3, ..., p_{n-1}, p_n) \tag{1}$$

where the cardinality of P is denoted as: $\mid P \mid = n$

Our goal is to produce a set of *diverse* workspace paths. We define *diversity* between paths by using a distance metric that measures how far is a path with respect to another paths in the set. This metric is defined in section 3.4.

3.2 Vertex-Vertex Distance (d_{min})

Given the vertices v_a and $v_b \in \mathcal{V}$, we define the distance between them (d_{min}) as the shortest path in $\mathcal{V}$ that connects both vertices. We choose this metric instead of Euclidean distance since it is more informative of the real distance between two vertices in the search space.

3.3 Vertex-Path Distance (d_P)

Consider a path P and a vertex $x \notin P$. The distance between x and P (d_P) is expressed as:

$$d_P(x, P) = \min_{p_i \in P} \{d_{min}(x, p_i)\} \tag{2}$$

Informally, d_P is the shortest of all the d_{min} between x and every vertex in P

3.4 Path-Path(s) Distance (D_P)

Given two paths P_A and P_B, we define the distance of P_A *with respect to* P_B as:

$$D_P(P_A, P_B) = \sum_{i=1}^{|P_A|} d_P(p_{Ai}, P_B) \tag{3}$$

Since we will be dealing with more an arbitrary number of paths, we further define the distance between a single path and a set of paths. Given $\mathcal{P}_k$ as:

$$\mathcal{P}_k = \bigcup_{i=1}^{k} P_i = P_1 \cup P_2 \cup ... \cup P_k \tag{4}$$

The distance between a path P_{k+1} and a set of paths $\mathcal{P}_k$ is:

$$D_P(P_{k+1}, \mathcal{P}) = \sum_{i=1}^{|P_{k+1}|} d_P(p_{k+1_i}, \mathcal{P}_k) \tag{5}$$

Equation 5 is a generalization of Equation 3 considering P_B to be the union of multiple paths. It must be noted that this metric is not necesssarily commutative, so in general:

$$D_P(A, B) \neq D_P(B, A).$$

This is why we define $\mathcal{P}$ as a sequence of paths.

Now we have all the needed definitions to enunciate the *diverse workspace path problem*:

4 Diverse Workspace Path Problem

Given a high dimensional manipulator with its end-effector in a starting workspace position v_{start}, the task is to find an executable workspace path that translates the end effector to the target location v_{target}. To this end, we need to generate a sequence of k workspace paths:

$$\mathcal{P} = (P_1, P_2, P_3..., P_k)$$

such that they hold the following general requirements:

- All paths considered cannot be arbitrarily long
- Each path $P_i \in \mathcal{P}$ maximizes its distance D_P with respect to its predecessors in $\mathcal{P}$. Equivalently:

$$P_i = \arg\max_{Q \in \mathcal{Q}} \{D_P(Q, \{P_1, ...P_{i-1}\})\} \tag{6}$$

  where $\mathcal{Q}$ represents the domain of all possible paths that join v_{start} and v_{target}. This condition intends that each path P_i is as far as possible from the existing paths. Since we generate the paths sequentially, it makes sense to define the distance metric only with respect to the paths already generated (predecessors).

In the next section, we present an algorithm to build $\mathcal{P}$.

5 Proposed Algorithm

Our goal is to generate k diverse paths such that they maximize the distance D_P between them while not growing arbitrarily long. We can express both these requirements as an added cost. Given a path P_{k+1}, its cost would be:

$$Cost(P_{k+1}) = Cost_D(P_{k+1}) + Cost_L(P_{k+1}) \tag{7}$$

where $Cost_D$ is a measure of the D_P between P and the other existing paths. We wish to maximize the distance between paths, hence vertices with high distance should have low costs. Considering this, we define $Cost_D$ as:

$$Cost_D(P_{k+1}) = \sum_{j=1}^{|P_{k+1}|} \{D_{max} - d_P(p_{i+1\,j}, \mathcal{P}_k)\} \tag{8}$$

where D_{max} is the maximum d_P of all vertices. This is added in order to make sure that $Cost_D$ is non-negative. $\mathcal{P}_k$ is the set of paths already generated.

$Cost_L$ is defined such that it penalizes path length. The longer the path, the higher this cost is:

$$Cost_L(P_{i+1}) = \alpha |P_{i+1}| \tag{9}$$

where the parameter α represents the length cost between two vertices. α is a tunable parameter to control the ratio between the path cost and the length cost.

Replacing (8) and (9) in (7) we obtain the final expression of path cost for our algorithm:

$$Cost(P_{k+1}) = \sum_{j=1}^{|P_{k+1}|} \{D_{max} - d_P(p_{k+1\,j}, \mathcal{P}_k) + \alpha |P_{k+1}| \tag{10}$$

In our implementation α and k (number of paths to be generated) are input parameters to our algorithm.

5.1 Workspace Implementation

Our algorithm generates a sequence of paths $\mathcal{P}$, in which each P_{k+1} maximizes its D_P with respect to the existing generated paths ($\mathcal{P}_k$). Therefore, at the highest level, our algorithm can be expressed as a loop which generates the next path in the sequence (P_k) at iteration k. This is shown in Algorithm 1 (FINDPATHSSEQUENCE):

Algorithm 1 produces a sequence $\mathcal{P}$ of paths sequentially. At each iteration a path P_i is generated by FINDPATH, which implements an A^* search through the search space (as defined in 3.1).

Algorithm 1: FindPathsSequence

Input: $\mathcal{V}$, $\mathcal{E}$, k, v_{start}, v_{target}, α
Output: $\mathcal{P}$

```
P = ∅ ;
ResetSearch();
for i ← 1 to k do
    P_i = FindPath(v_start, v_target, α) ;
    P .add(P_i) ;
    ResetSearch();
    UpdateDistanceCosts(P) ;
return P ;
```

From Eq.(7) we can derive the cost of each vertex v to be used in FINDPATH:

$$Cost(v) = (v.\text{COSTDISTANCE} + \alpha)$$

where $v.\text{COSTDISTANCE} = (D_{max} - d_P(v, \mathcal{P}_i))$ (Eq 8). d_P is the shortest path from v to $\mathcal{P}_i$. Algorithm 3 (PATHDISTANCETRANSFORM) calculates d_P for all $v \in \mathcal{V}$ with respect to $\mathcal{P}_k$ and D_{max} in each iteration.

PATHDISTANCETRANSFORM is a naive implementation of the fairly common *Distance Transform* ([4], [17]), which, given as input a set of points X and a subset Y, calculates the shortest distance between all $x \in X$ and Y (analogous to applying d_P to all $v \in \mathcal{V}$).

To summarize, our algorithm does the following:

1. Generates a path P_{k+1} that minimizes the path distance cost between P_{k+1} and the already generated $\mathcal{P}_k$ (FINDPATH).
2. Adds P_{k+1} to $\mathcal{P}_k$
3. Resets the search parameters used and sets the *costDistance* of all vertices to zero
4. Calls UPDATEDISTANCETRANSFORM, which invokes PATHDISTANCETRANSFORM to update the distance of all vertices with respect to $\mathcal{P}_k$, stores D_{max}, and updates the distance costs of each node.
5. Repeats k times

Notice that for P_1, $\mathcal{P} = \emptyset$ and $v.\text{COSTDISTANCE} = 0$ since there are no previous paths generated . Thus, the only cost per vertex is the length cost. This is equivalent to saying that the first path generated P_1 is the shortest path. For the following $k-1$ paths, $\mathcal{P}$ will be updated with the paths generated at each iteration.

Algorithm 2: `UpdateDistanceCosts`

Input: $\mathcal{V}, \mathcal{P}$

Output: Updated distance costs from every $v \in \mathcal{V}$ w.r.t. $\mathcal{P}$

```
1 PathDistanceTransform(V, P) ;
2 D_MAX ← arg max_{v_i ∈ V}{v_i.distance} ;
3 forall v ∈ V do
4     v.costDistance ← D_MAX − v.distance ;
```

5.2 Mapping from Workspace to Jointspace

Section 5.1 explained how to obtain a sequence of workspace paths for the end effector. This section refers to the translation of these paths to jointspace, which is the final input for our manipulator.

Forward kinematics gives the end-effector position as a function of the degrees of freedom of the manipulator (θ):

$$x_{ee} = f(\theta) \tag{11}$$

In general there may not be a solution for 11, i.e. in the case in which x_{ee} is out of the reachable space of the manipulator. Or, in the case of redundant manipulators, there may be no unique solution.

A workspace path consists of a sequence of cartesian coordinates in 3D. There are options to map this path, depending of the type of manipulator. For non-redundant manipulators the mapping can be done using analytical inverse kinematics (IK). In the case of redundant manipulators, IK may also be used with a search of the redundant parameter space. Another alternative in both cases is mapping through the robot Jacobian.

In our implementation with a simulated 7-DOF robotic arm, we used the Jacobian pseudo-inverse with the Damped Least Square Method in order to work robustly in presence of singularities ([3]). So, having a particular arm configuration θ with its end effector in x and mapping the location x_r, the update rule for θ

Algorithm 3: `PathDistanceTransform`

Input: $\mathcal{V}$, $\mathcal{P}$
Output: v with distance values updated w.r.t. $\mathcal{P}$, $\forall v \in \mathcal{V}$

```
1  forall v ∈ V do
2      v.distance = ∞ ;
3  forall v ∈ P do
4      v.distance = 0 ;
5  Queue ← P ;
6  TempQueue ← ∅ ;
7  while Queue ≠ ∅ do
8      forall q ∈ Queue do
9          for all n ∈ q.neighbors do
10             newDist ← q.distance + EdgeCost(q,n) ;
11             if newDist < n.distance then
12                 n.distance ← newDist ;
13                 TempQueue.pushBack(n) ;
14     Queue ← TempQueue ;
15     TempQueue ← ∅ ;
```

is expressed as:

$$\Delta x = x_r - x \tag{12}$$

$$J^{\dagger}(\theta) = J^T(\theta)(J(\theta)J^T(\theta) + \lambda I)^{-1} \tag{13}$$

$$\Delta\theta = J^{\dagger}(\theta)\Delta x \tag{14}$$

$$\theta = \theta + \Delta\theta \tag{15}$$

After generating a set of diverse workspace paths, we proceed to do the mapping of the paths in ascending order. As soon as the path evaluated is successfully mapped and its corresponding jointspace path generated, we declare it as a solution: the rest of workspace paths are not evaluated since a solution was found; otherwise, our manipulator considers the next path in the sequence of paths generated. This allows us to have more than one alternative, in case one of the first paths cannot be executed.

5.3 Additional Details

For the implementation shown in the accompanying video we used a damping factor $\lambda = 0.01$. Also, the discretization of the environment was based on a voxel unit of $0.02 \times 0.02 \times 0.02m$. During the discretization we used a padding around the obstacles to account for the radius of the end effector.

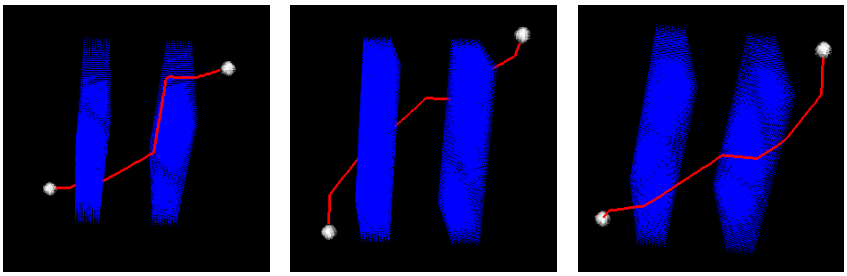
6 Experiments and Results

We implemented the algorithm proposed in Section 5 and tested it in manually generated environments in order to observe and evaluate the results. All the examples presented in this section consisted of a $80 \times 80 \times 80$ voxelized space (around half a million of vertices) populated with box-shaped obstacles for ease of visualization. The results presented here focus primarily in the generation of the workspace paths and their diversity. Two applications of these workspace paths and their mapping to jointspace are shown in the accompanying video. All experiments were run on an Intel Core i7 (1.6GHz).

The explanations in this section will focus on 4 test cases, shown in Figures 3, 4, 5 and 6. Table 1 shows the main characteristics of these four environments. The figures only show the first paths generated for space constraint reasons.

Our algorithm produces k paths, where k is an input to the planner. Both k and α parameters used in each case are also in Table 1.

As we mentioned in Section 5, we can consider α as a measure of the length cost with respect to the distance cost. Higher values of α restrict more strictly the



(a) Path 1 (b) Path 2 (c) Path 3

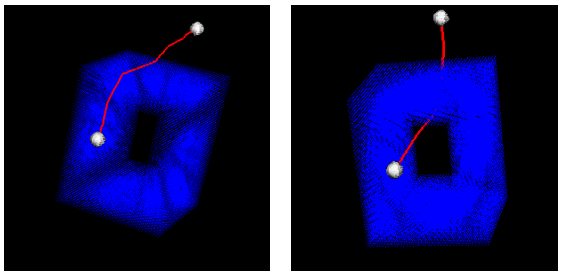
Figure 3: Experiment 1 with $\alpha = 0.25$. The path depicted in Fig.3(a) is the shortest one. Path 3(b) and 3(c) are the next found

Table 1: Experimental Setups specifications

Setup	***Obstacles***	***Homotopic classes***	***k***	α
Setup 1	2	1	5	0.25
Setup 2	4	2	5	0.01
Setup 3	7	1	5	0.01
Setup 4	1	1	5	0.01

length of the generated paths, whereas a low α favours longer paths with larger D_P distance. In our experiments, we have found that an $\alpha \in [0.01, 0.3]$ (and normalizing the distance cost between 0 and 1) *usually* yield paths that are diverse but not much longer than the shortest path.

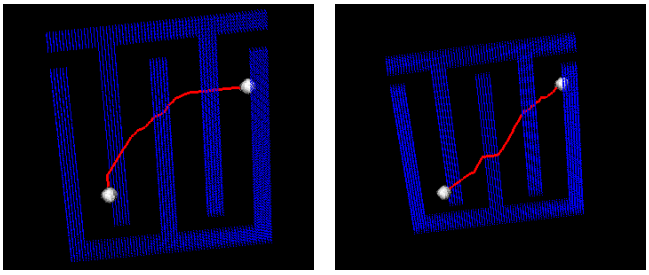
Finally, Table 2 and 3 show some statistics related to computation time between iterations (for all 4 experiments we generated $k = 5$ paths. In the table we shows the results of iterations $2 - 5$). Table 2 shows that the planner spends considerably more time in calculating and updating the distance from the vertices to the path set (PATHDISTANCETRANSFORM and UPDATEDISTANCECOSTS) with respect to the search time (FINDPATH). However, it should be noted that our algorithm currently uses a naive implementation of the Distance Transform, which is much slower compared to other, far more efficient methods (e.g. [5]) which generally operate in a regular discretized space. In our case, our search space is not a regular voxelized grid (remember that we only consider the free voxels, hence the search space is the union of free space regions) so some modifications to the original DT algorithm is required. Table 3 shows a comparison of the search time between iterations for each of the 4 scenarios. As we can see, the time is nearly constant between iterations and between experiments.



(a) Path 1 (b) Path 2

(c) Path 3 (d) Path 4

Figure 4: Experiment 2 with $\alpha = 0.01$. 4 first generated paths

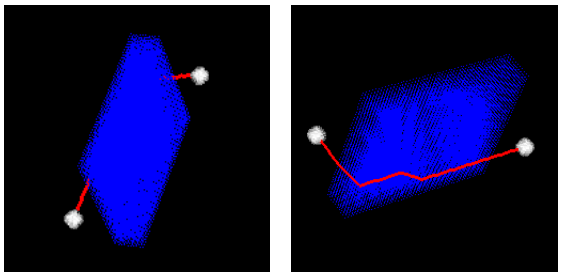


(a) Path 1 (b) Path 2

Figure 5: Experiment 3 with $\alpha = 0.01$. Two first generated paths

Table 2: Main Distribution of Computation Time

Setup	***% Search time***	***% Update distance time***
Setup 1	19.7	80.3
Setup 2	22.9	77.1
Setup 3	23.1	76.9
Setup 4	19.8	80.2



(a) Path 1 (b) Path 2

Figure 6: Experiment 4 with $\alpha = 0.01$. 2 first generated paths

Table 3: Search time per iteration. The search time for the first path is not considered.

Setup	***Search time per iteration***			
	1	**2**	**3**	**4**
Setup 1	1.290s	1.210s	1.220s	1.260s
Setup 2	1.280s	1.200s	1.240s	1.240s
Setup 3	1.500s	1.360s	1.380s	1.270s
Setup 4	1.230s	1.310s	1.270s	1.420s

7 Conclusions and Future work

We have presented a simple and intuitive planning algorithm that generates a sequence of k diverse workspace paths in 3D static discrete environments and introduced the alternative concept of *diversity* applied to a sequence of paths, such that we maximize a distance metric between any path and its predecessor. We use a parameter α to represent the ratio between the distance cost and the length cost of a path. Our main motivation in generating diverse workspace paths is to provide a manipulator with more flexibility to select which path to follow to accomplish a specified task involving reaching an end-effector location.

Currently, we are investigating methods to automatically tune the parameter α based on discrete optimization approaches. We are also exploring the possibility of using different metrics to express the *diversity* of a sequence of paths. Finally, we are interested in the particular problem of workspace-jointspace mapping and how it can affect the selection of workspace paths.

References

[1] Subhrajit Bhattacharya, Maxim Likhachev, and Vijay Kumar. Search-Based Path Planning with Homotopy Class Constraints. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[2] Subhrajit Bhattacharya, Maxim Likhachev, and Vijay Kumar. Identification and Representation of Homotopy Classes of Trajectories for Search-based Path Planning in 3D. In *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA, June 2011.

[3] Samuel R. Buss. Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares Methods. Technical report, IEEE Journal of Robotics and Automation, 2004.

[4] R. Fabbri, L. Costa, J. Toreni, and O. Bruno. 2D Euclidean Distance Transform Algorithms: A Comparative Survey. *Computing Surveys*, 40:2–44, 2008.

[5] Pedro Felzenszwalb and Daniel Huttenlocher. Distance Transforms of Sampled Functions. Technical Report TR2004-1963, Cornell Computing and Information Science Technical Report, 2004.

[6] Colin Green and Alonzo Kelly. Toward Optimal Sampling in the Space of Paths. In *13th International Symposium of Robotics Research*, November 2007.

[7] Dima Grigoriev and Anatol Slissenko. Polytime Algorithm for the Shortest Path in a Homotopy Class Amidst Semi-Algebraic Obstacles in the Plane. In *The International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 17–24, 1998.

[8] Peter Hart, Nils J. Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[9] John Hershberger and Jack Snoeyink. Computing Minimum Length Paths of a Given Homotopy Class. *Computational Geometry: Theory and Applications*, 4:331–342, 1991.

[10] Takeo Igarashi and Mike Stilman. Homotopic Path Planning on Manifolds for Cabled Mobile Robots. In *Proceedings of the The Ninth International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2010.

[11] Leonard Jaillet and Thierry Siméon. Path deformation roadmaps: Compact graphs with useful cycles for motion planning. *International Journal of Robotic Research*, 27(11-12):1175–1188, 2008.

[12] Lydia E. Kavraki, Petr Svestka, Jean-claude Latombe, and Mark H. Overmars. Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Transactions on Robotics and Automation*, 12:566–580, 1996.

[13] Ross Alan Knepper, Siddhartha Srinivasa, and Matthew T. Mason. Toward a deeper understanding of motion alternatives via an equivalence relation on local paths. *International Journal of Robotics Research*, 31(2):168–187, February 2012.

[14] James J. Jr. Kuffner and Steven M. Lavalle. RRT-connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 995–1001, 2000.

[15] Anthony Maciejewski and Charles Klein. Obstacle Avoidance for Kinematically Redundant Manipulators in Dynamically Varying Environments. In *The International Journal of Robotics Research*, volume 4, 1985.

[16] E. Schmitzberger, J.L. Bouchet, M. Dufaut, M. Wolf, and R. Husson. Capture of Homotopy Classes with Probabilistic Roadmap. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[17] Daniel E. Whitney. Resolved Motion Rate Control of Manipulators and Human Prostheses. In *IEEE Transactions on Man-Machine Systems*, volume 2, pages 47–53. 1969.